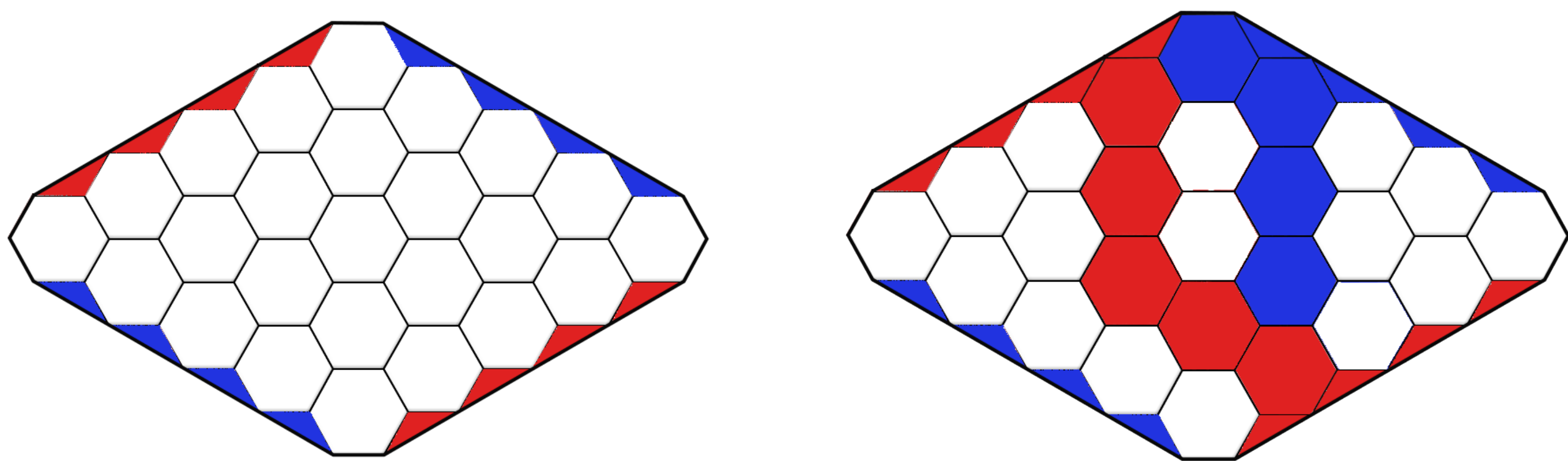


Evolving a Hex-Playing Agent

Michael McCarver, Dr. Rob LeGrand

Abstract

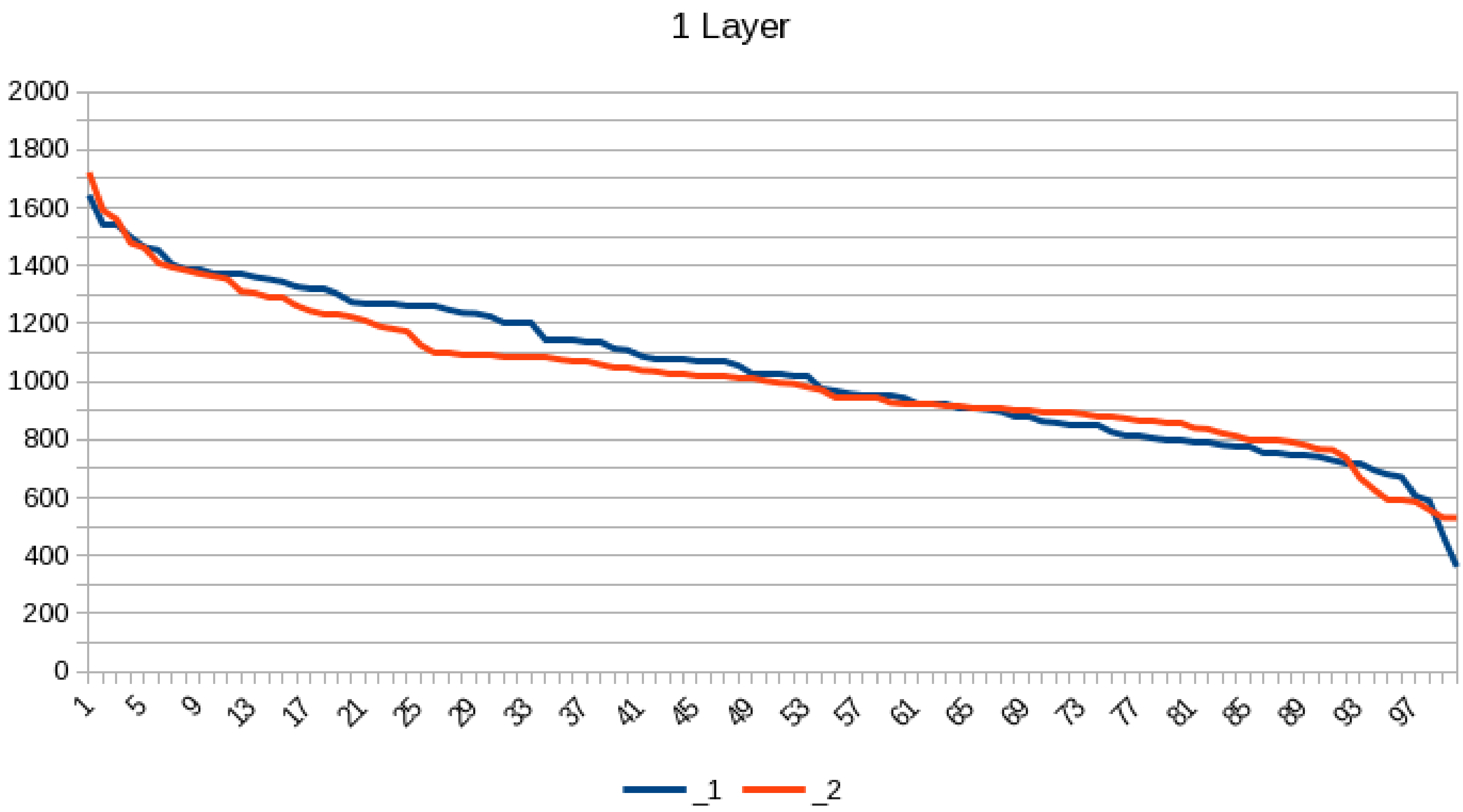
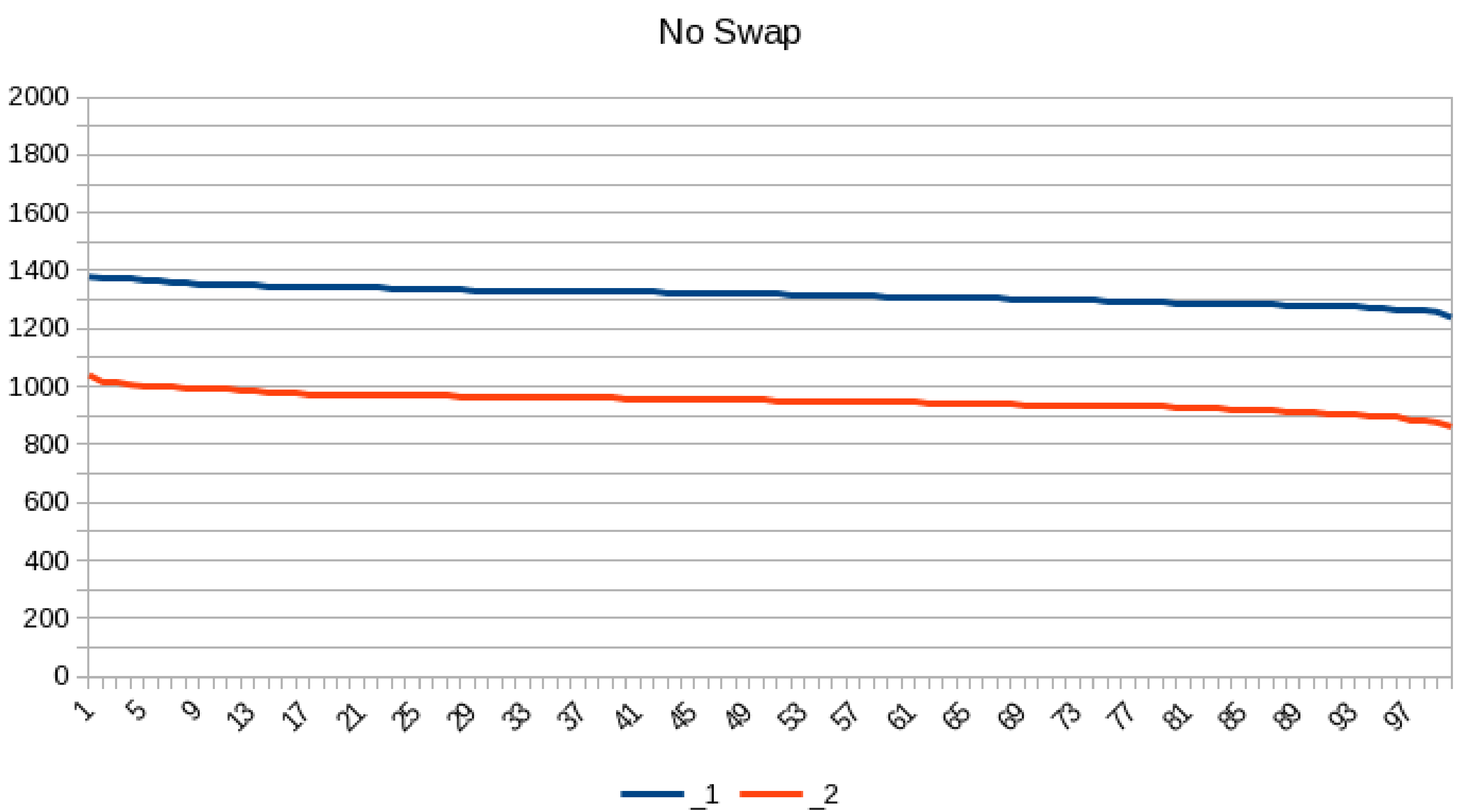
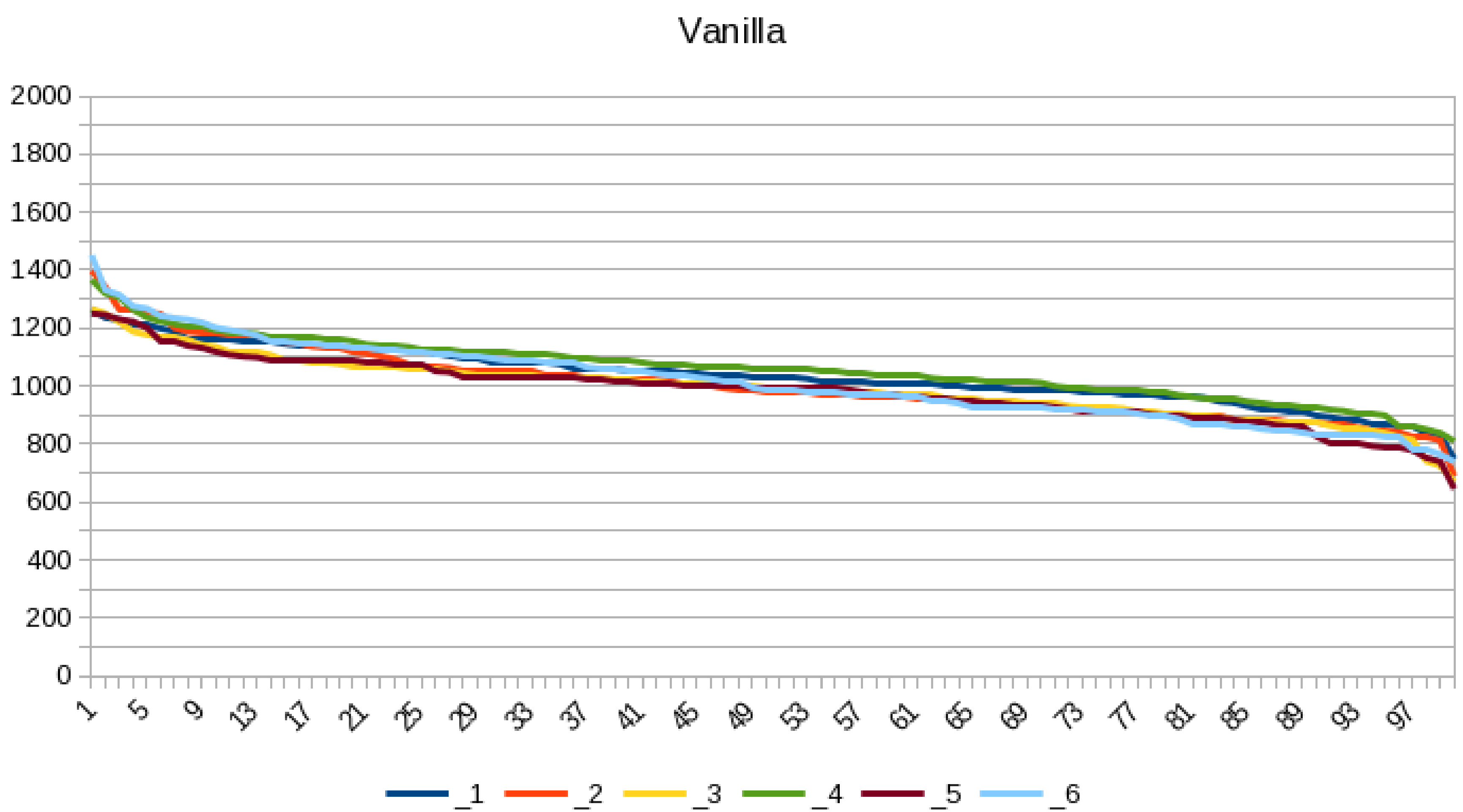
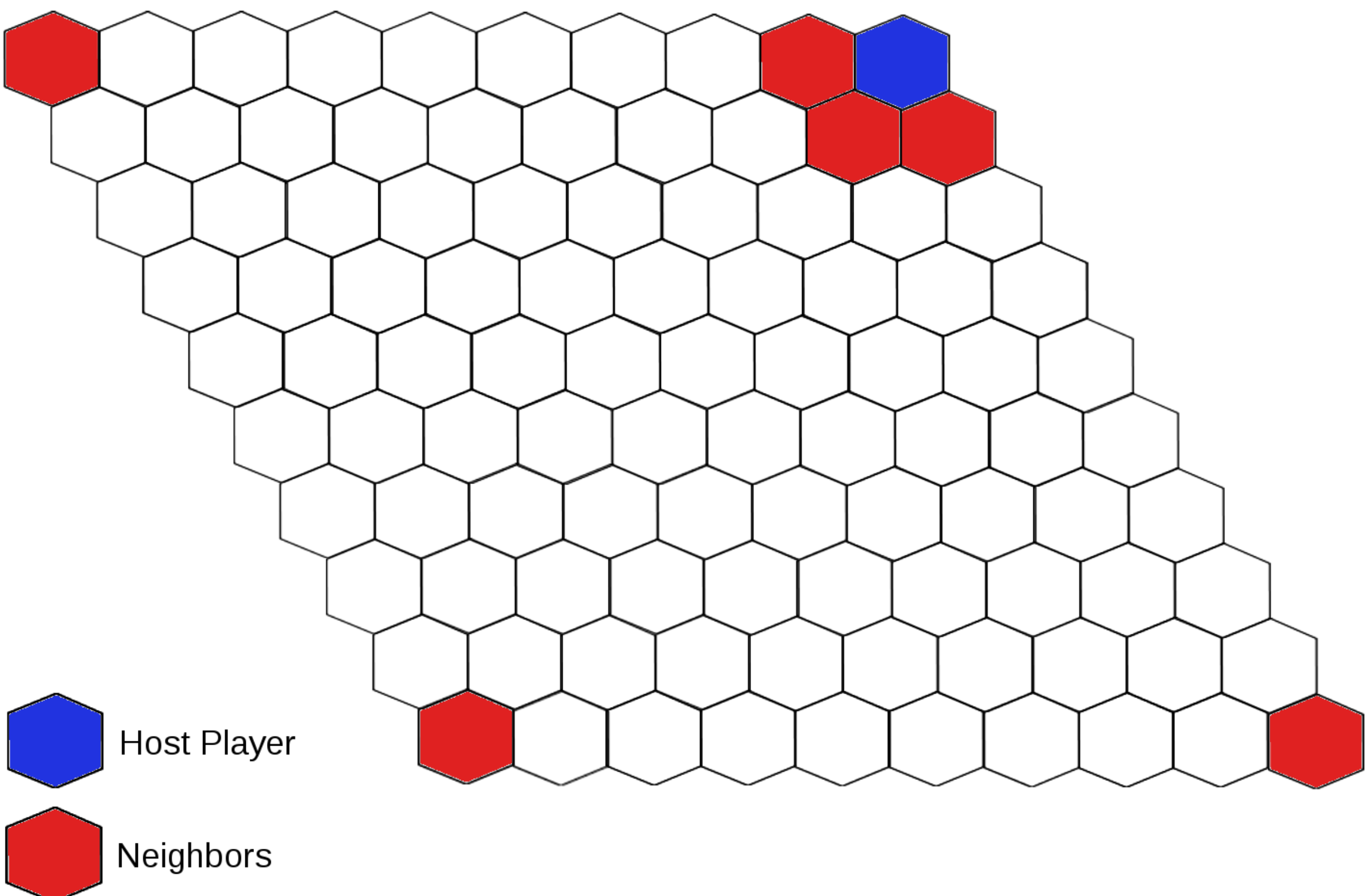
Hex is a two-player adversarial board game in which there is always exactly one winner. Although it is known that a winning strategy exists for the first player, such strategies are difficult to find due to the large branching factor of Hex's game trees. A subset of Artificial Intelligence research is devoted to optimizing search algorithms, such as Minimax, pursuant to searching these game trees and solving Hex boards for any game position. Our research is not concerned with perfect playing strategies. Instead of Minimax approaches, we use Artificial Neural Networks and Genetic Algorithms to test the bounds of how quickly and how effectively Artificial Neural Networks are able to learn to evaluate board-states of a game. We experiment with network topology and evolution strategies and compare different approaches using metrics we developed.



Sample game board and completed game (red player wins)

Setup

A Hex-playing agent is a program that uses an Artificial Neural Network (ANN) to heuristically evaluate a Hex board by processing the state of the board (locations of moves played by either player). We create a population of 100 Hex-players with randomized genes, which here means randomized weight values within the ANN. The population is arranged in a 10x10 hexagonal torus, so that each player has 6 neighbors, and players at the 'top' and 'bottom' as well as players on the 'left' and 'right' of the map neighbor each other.



Method

A single iteration of execution involves many steps:

- 1) For each player, play 2 games against each neighbor. Keep track of the number of games each player wins (for use in a fitness function).
- 2) After all games have been played, iterate through each player's genes, deciding whether to keep them or to replace them with a neighbor's. If the decision is to replace the gene, choose one neighbor's gene according to a weighted probability. The weights for the probability are determined by our fitness function, therefore players that win often are more likely to have their genes chosen.
- 3) Mutate each gene slightly. By mutate we mean to change the value of the weight. We choose a new value for the weight according to a normal distribution centered on the original value of the weight.
- 4) After each gene has been mutated and/or bred, swap (or copy) two random weights within the ANN. This step was not included in every experiment we ran. Notably, the 'No Swap' experiments charted did not include this step.

Experimentation/Results

In order to have a basis for comparison, we arbitrarily designated a specific arrangement of network topology and genetic evolution strategies to act as control. We ran this experiment 6 times and named the populations collectively 'Vanilla'. The Vanilla specifications are:

Network Topology: 5 input nodes leading to 1 output node

Swap Strategy: Always swap weights exactly once

Inertia: Always choose a player's gene according to the fitness function

Fitness Function: Probability of choosing a player's gene is $\frac{\text{number of games won by player}}{\text{total games won among neighbors}}$

Each additional experiment differed from Vanilla by one specification and was run twice (iterations have a high time cost, so we were limited in the number of iterations we could run).

We created a number of metrics to compare experiments with each other. To the left we have charts which graphically represent one of our metrics. In this metric, each of the 100 players in a population play 2000 games against completely random players (1000 each as first and second player). The x-axis is the ranking of players within the population and the y-axis is the total games won by that player.

Conclusions

The Vanilla populations have tightly distributed curves, which seems to imply that the strategy produces consistent populations. In contrast, the No Swap strategy (in which weights are never swapped within a population) creates flat, separate curves. This seems to imply the strategy creates homogeneous players at inconsistent playing ability. The 1 Layer strategy (with a simple, single-node model) seems to consistently produce the best and worst players we have.

None of our evolved agents planned ahead more than one move, so they aren't strong enough Hex players to defeat a decent human opponent. But our comparisons of approaches found clear patterns that may improve future attempts at creating AI for Hex.